C++ fundamentals

THURSDAY, APRIL 10

Section materials curated by Jonathan Coronado, Yasmine Alonso, and Sean Szumlanski, drawing upon materials from previous quarters

This week’s section exercises focuses on exploring some C++ quirks and learning about strings. Have fun!

Each week, we will also be releasing a Qt Creator project containing starter code and testing infrastructure for that week's section problems. When a problem name is followed by the name of a **.cpp** file, that means you can practice writing the code for that problem in the named file of the Qt Creator project. Here is the zip of the section starter code:

📦 Starter project

1. References Available Upon Request

Topics: Reference parameters, range-based for loops

Reference parameters are an important part of C++ programming, but can take some getting used to if you’re not familiar with them. Trace through the following code. What does it print?

```
void maui(string s) {
    for (int i = 0; i < s.length(); i++) {
        s[i] += 2;
    }
}

void sina(string& s) {
    for (int i = 0; i < s.length(); i++) {
        s[i] += 2;
    }
}

void moana(string& s) {
    for (char ch : s) {
        ch += 2;
    }
}

void heihei(string& s) {
    for (char& ch : s) {
        ch += 2;
    }
}

string teFiti(string& s) {
    string result;
    for (char ch : s) {
        result += (ch + 2);
    }
    return result;
}

int main() {
    string s = "umm";

    maui(s);
    cout << s << endl;

    sina(s);
    cout << s << endl;

    moana(s);
    cout << s << endl;

    heihei(s);
    cout << s << endl;

    teFiti(s);
    cout << s << endl;

    return 0;
}
```

Solution

Here's the output from the program:

```
umm
woo
woo
yqq
yqq
```

Here's a breakdown of where this comes from:

- The **maui** function takes its argument by value, so it's making changes to a copy of the string, not the original string itself. That means that the values are unchanged back in main.
- The **sina** function, on the other hand, takes its argument by reference and loops through the string directly, so the original string is modified back in main.
- The **moana** function uses a range-based for loop to access the elements of the string. Even though the string is passed by reference, the loop makes a copy of each character of the string, so the changes made in the loop only change the temporary copies of the characters and not the characters of the string. That makes the string remain unchanged back in main.
- **heihei**, on the other hand, uses char& as its type for the range-based for loop, so in a sense it's really iterating over the elements of the underlying string. Therefore, its changes stick.
- The **teFiti** function creates and returns a new string with a bunch of updated values, but the return value isn't captured back in main.

2. Pig-Latin (piglatin.cpp)

Topics: Strings, reference parameters, return types

Write two functions, **pigLatinReturn** and **pigLatinReference**, that accept a string and convert said string into its pig-Latin form. To convert a string into pig-Latin, you must follow these steps:

- Split the input string into 2 strings: a string of characters BEFORE the first vowel, and a string of characters AFTER (and including) the first vowel.
- Append the first string (letters before the first vowel) to the second string.
- Append the string "ay" to the resulting string.

Here are a few examples…

`sean -> eansay`

`jonathan -> onathanjay`

`julian -> ulianjay`

You will need to write this routine in two ways: once as a function that returns the pig-Latin string to the caller, and once as a function that modifies the supplied parameter string and uses it to store the resulting pig-Latin string. These will be done in **pigLatinReturn** and **pigLatinReference**, respectively. You may assume that your input is always a one-word, all lowercase string with at least one vowel.

Here's a code example of how these functions differ…

```
string name = "jonathan";
string str1 = pigLatinReturn(name);
cout << str1 << endl; // prints "onathanjay"

pigLatinReference(name);
cout << name << endl; // prints "onathanjay"
```

Once you've written these functions, **discuss with your section the benefits and drawbacks of these two approaches**. Which do you feel is easier to write? Which do you think is more convenient for the caller? Do you think one is better style than the other?

Solution

```
// Use const because VOWELS won't change -- no need to declare
repeatedly
// in isVowel.
const string VOWELS = "aeiouy";

// Helper function, which I'd highly recommend writing!
bool isVowel(char ch) {
    // A little kludgy, but the handout guarantees that
    // ch will ALWAYS be lower case :)
    // NOTE: For an assignment, you probably want a more robust
isVowel.
    return VOWELS.find(ch) != string::npos;
}

string pigLatinReturn(string input) {
    int strOneIndex = 0;
    for (int i = 0; i < input.length(); i++) {
        if (isVowel(input[i])) {
            strOneIndex = i;
            break;
        }
    }
    string strOne = input.substr(0, strOneIndex);
    string strTwo = input.substr(strOneIndex);
    return strTwo + strOne + "ay";
}

void pigLatinReference(string &input) {
    int strOneIndex = 0;
    for (int i = 0; i < input.length(); i++) {
        if (isVowel(input[i])) {
            strOneIndex = i;
            break;
        }
    }
    string strOne = input.substr(0, strOneIndex);
    string strTwo = input.substr(strOneIndex);
    input = strTwo + strOne + "ay";
}
```

Notice how similar these two approaches are – the only difference is how the result is handled at the very end. To address the discussion questions, although the **`pigLatinReference`** function is marginally more efficient because it doesn't need to make a copy of the input string, **`pigLatinReturn`** is probably more intuitive for both the caller and the writer: if the function's job is to somehow output some product, returning is the most explicit way to do so. In that way, a function that returns is also better style – it's makes the purpose of the function clearer to the reader.

If you wanted to combine the efficiency of **`pigLatinReference`** with the clarity of **`pigLatinReturn`**, I would recommend writing a function that takes in the input string by **`const reference`**, basically

```
string pigLatin(const string &input);
```

Although the const isn't explicitly necessary, it's nice to have because you never need to modify input. Moreover, you still get the efficiency gains from pass-by-reference while also writing very-understandable code.

3. Program analysis: C++isms you should know (programanalysis.cpp)

Topics: Types, References, range based loops, strings, stanford C++ library

In the following, we will analyze a simple program that filters last names whose end match a specific substring. Given an input string of format:

```
name1,name2, ...
```

and a string suffix, the program returns all the names in the input string that ends with the suffix.

```
#include "SimpleTest.h"
#include "vector.h"
#include "strlib.h"

using namespace std;

/*
   @param input: input string whose last names will be filtered
   @param suffix: the substring which we will filter last names by
   Functionality: this function filters the input string and returns last
names
        that end with 'suffix'
*/
Vector<string> filter(string input, string suffix)
{
    Vector<string> filteredNames;
    Vector<string> names = stringSplit(input, ',');

    for (string name: names) {
        // convert to lowercase so we can easily compare the strings
        if (endsWith(toLowerCase(name), toLowerCase(suffix))) {
            filteredNames.add(name);
        }
    }
    return filteredNames;
}

STUDENT_TEST("Filter names") {
    Vector<string> results = filter("Zelenski,Szumlanski,Alonso", "Ski");
    EXPECT_EQUAL(results, {"Zelenski","Szumlanski"});

    results = filter("AmbaTi,Szumlanski,Tadimeti", "TI");
    Vector<string> expected = {"AmbaTi", "Tadimeti"};
    EXPECT(results == expected);

    results = filter("Zelenski,Szumlanski,Alonso", "nso");
    EXPECT_EQUAL(results, {"Alonso"});

    results = filter("Szumlanski,Coronado", "AaS");
    EXPECT_EQUAL(results, {});

    // what other tests could you add?
}
```

Solution

Another useful test could be this:

```
results = filter("Zelenski,Szumlanski,Rodriguez Cardenas", "uez");
EXPECT_EQUAL(results, {});
```

Since the input string is split by commas, "Rodriguez Cardenas" is processed as one string, which does not end on "uez".

4. Returning and printing

Topics: Function call and return, return types

Below is a series of five **printLyrics_v#** functions, each of which has a blank where the return type should be. For each function, determine

- what the return type of the function should be,
- what value, if any, is returned, and
- what output, if any, will be produced if that function is called.

Is it appropriate for each of these functions to be named **printLyrics**? Why or why not?

```
_____ printLyrics_v1() {
    cout << "Havana ooh na na" << endl;
}
_____ printLyrics_v2() {
    return "Havana ooh na na";
}
_____ printLyrics_v3() {
    return "H";
}
_____ printLyrics_v4() {
    return 'H';
}
_____ printLyrics_v5() {
    cout << 'H' << endl;
}
```

Solution

```
void printLyrics_v1() {
    cout << "Havana ooh na na" << endl;
}

string printLyrics_v2() {
    return "Havana ooh na na";
}

string printLyrics_v3() {
    return "H";
}

char printLyrics_v4() {
    return 'H';
}

void printLyrics_v5() {
    cout << 'H' << endl;
}
```

The function **printLyrics_v1** doesn't return anything - it just sends information to the console. As a result, its return type should be void. Since this function prints out the string "Hakuna Matata", the name "printLyrics" is appropriate!

The functions **printLyrics_v2** and **printLyrics_v3** both return strings, since C++ treats anything in double-quotes as a string. Neither of these functions actually prints the string though, which means that the name "printLyrics" is inappropriate for both functions.

The function **printLyrics_v4** returns a char, since C++ treats anything in single-quotes as a character. This function also does not print anything, which means that the name "printLyrics" is inappropriate.

Finally, the function printLyrics_v5 doesn't return anything, which means its return type is void. This function does print out a char. The name "printLyrics" is still inappropriate because only a single character is being printed from this function, not an entire line of lyrics.

5. Debugging Deduplicating (deduplicate.cpp)

Topics: Vector, strings, debugging

Consider the following **incorrect** C++ function, which accepts as input a **`Vector<string>`** and tries to modify it by removing adjacent duplicate elements:

⚠️ **WARNING!** The following function is buggy. 🐜

```
void deduplicate(Vector<string> vec) {
    for (int i = 0; i < vec.size(); i++) {
        if (vec[i] == vec[i + 1]) {
            vec.remove(i);
        }
    }
}
```

The intent behind this function is that we could do something like this:

```
Vector<string> hiddenFigures = {
    "Katherine Johnson",
    "Katherine Johnson",
    "Katherine Johnson",
    "Mary Jackson",
    "Dorothy Vaughan",
    "Dorothy Vaughan"
};

deduplicate(hiddenFigures);
// hiddenFigures = ["Katherine Johnson", "Mary Jackson", "Dorothy Vaughan”]
```

The problem is that the above implementation of **deduplicate** does not work correctly. In particular, it contains three bugs. First, find these bugs by writing test cases that pinpoint potentially erroneous situations in which the provided code might fail, then explain what the problems are, and finally fix those errors in code.

Solution

There are three errors here:

1. Calling **`.remove()`** on the **`Vector`** while iterating over it doesn't work particularly nicely. Specifically, if you remove the element at index **`i`** and then increment **`i`** in the for loop, you'll skip over the element that shifted into the position you were

previously in.

2. There's an off-by-one error here: when `i = vec.size() - 1`, the indexing `vec[i + 1]` reads off the end of the `Vector`.
3. The `Vector` is passed in by value, not by reference, so none of the changes made to it will persist to the caller.

Here's a corrected version of the code:

```
void deduplicate(Vector<string>& vec) {
    for (int i = 0; i < vec.size() - 1; ) {
        if (vec[i] == vec[i + 1]) {
            vec.remove(i);
        } else {
            i++;
        }
    }
}
```

[Credit to Andrew Tierno for the alternate solution] Alternatively, you can also re-write the function to use a loop that traverses the vector from right-to-left, which is a common pattern when working with deleting items from linear collections. A solution that does so could look like this:

```
void deduplicate(Vector<string>& vec) {
    for (int i = vec.size() - 1; i > 0; i--) {
        if (vec[i] == vec[i - 1]) {
            vec.remove(i);
        }
    }
}
```

Setting up your environment correctly

Topics: QT Creator

- Navigate to **`Recommended Qt settings`** and set all recommended settings suggested. You'll use Qt Creator for all assignments this quarter, so it's important that you use the best settings to make you more efficient.
- Make a CS106B folder in your home directory. You can do so by:
  - Opening your finder(if you use a Mac) or Windows Explorer (if you use Windows)
  - Click on MacintoshHD for Mac or Primary Drive (C:) for windows
  - Click on Users
  - Click your name
  - Right click and make new folder.
  - Name the folder with ordinary characters, no spaces, special characters or emojis When you download new assignments and section materials, be sure to store them here. This will make it so that Qt has all the permissions it needs to run your programs
- If you use a Mac
  - Right click on this section's .pro file
  - Select Get info
  - Check that Qt Creator has been set as the default program to open .pro files. If not, choose Qt Creator from the drop down, and click on Change All
- If you use Windows
  - Open File Explorer (open any folder).
  - Click the View tab.
  - Select "File name extension" This will make it so you can see which files end with .pro

- Helpful Qt Creator hot keys (if you use windows, replace Command with Ctrl):
  - Command + B to build your program
  - Command + R to run your program
  - Command + Y to run in debug mode